Edinburgh Napier
UNIVERSITY

RESTFUL DESIGN FOR THE WEB

Web Tech
SET08101

Simon Wells
s.wells@napier.ac.uk
http://www.simonwells.org

TL/DR

- A lot more to designing effective dynamic web sites than just generating HTML on request

AIMS

- At the end of this (sub-section) of the topic you will:
  - Understand the relationship between APIs & the Web
  - Know about the REST architectural style
  - Be able to consider the design of a URL hierarchy in terms of the collections of data that it exposes

OVERVIEW

- Developing a dynamic site isn’t **just** executing code on the server when a request comes in
- There are principled approaches to designing dynamic sites
- Intimately connected with design of HTTP APIs
- Dynamic sites need not just return HTML (JSON, XML or any other **mediatype**)

APIS & THE WEB

- **A**pplication **P**rogramming **I**nterfaces
  - A user interface (but for different groups of users)
  - *Generally:* A way for communications to occur between software, i.e.
    - Web Server <——> Browser (HTML/CSS/JS)
    - Web Server <——> Mobile App (JSON\XML)
- An API can return data formatted in different ways depending upon the request

Edinburgh Napier
UNIVERSITY

WHY?

- Build platforms rather than sites
  - People do innovative things with what you provide
  - Contributes back more data
  - Interesting applications attract more users
    1. You don't have time to develop everything for everyone
    2. Restricting what users can do can alienate them

Which organisations have developed a platform rather than a site?

GOOGLE, NETFLIX, FLICKR, GITHUB, TWITTER, FACEBOOK, OTHERS….?

ONGOING DEBATE

- Even if you're not sharing a public API & building a platform API design is important

- A good, well-structured, easy to use API can help your site to be:

  scalable, extendable, easy to develop for, maintainable, robust - **all the things that we should aim for**

- But a nuanced, ongoing debate about how to achieve this - no official guidelines

REST

- **RE**presentational **S**tate **T**ransfer (**REST**)
- A software **architectural style** for the WWW
- Developed by Roy Fielding (architect of HTTP1.1 [96-99] with Berners-Lee) in his Ph.D Thesis (2000 "Architectural Styles and the Design of Network-based Software Architectures"
- A set of coordinated constraints on designing components within a distributed hypermedia system - *Aim for high-performance & maintainable architectures*
- If a system conforms to the constraints of REST then can be termed *RESTful* - however many APIs only implement part of the constraints - *buzzwordy*

REST's client–server separation of concerns simplifies component implementation, reduces the complexity of connector semantics, improves the effectiveness of performance tuning, and increases the scalability of pure server components. Layered system constraints allow intermediaries—proxies, gateways, and firewalls—to be introduced at various points in the communication without changing the interfaces between components, thus allowing them to assist in communication translation or improve performance via large-scale, shared caching. REST enables intermediate processing by constraining messages to be self-descriptive: interaction is stateless between requests, standard methods and media types are used to indicate semantics and exchange information, and responses explicitly indicate cacheability.

Fielding (2000)
Chapter 5 of "Architectural Styles and the Design of Network-based Software Architectures"

ARCHITECTURAL PROPERTIES

- The REST approach **affects** (ideally positively) a range of identifiable properties of distributed hypermedia systems:
  - (Perceived) Performance - component interactions can be a dominant factor in user perception of system performance and network efficiency
  - Scalability - Support large numbers of components & interactions between them
- Simple interfaces
- (Run-time) modifiability of interfaces
- Visibility of communication between components
- Portability of components (move code with data)
- Reliability - system should be resistant to failure even if components, connectors, or data fail in some way

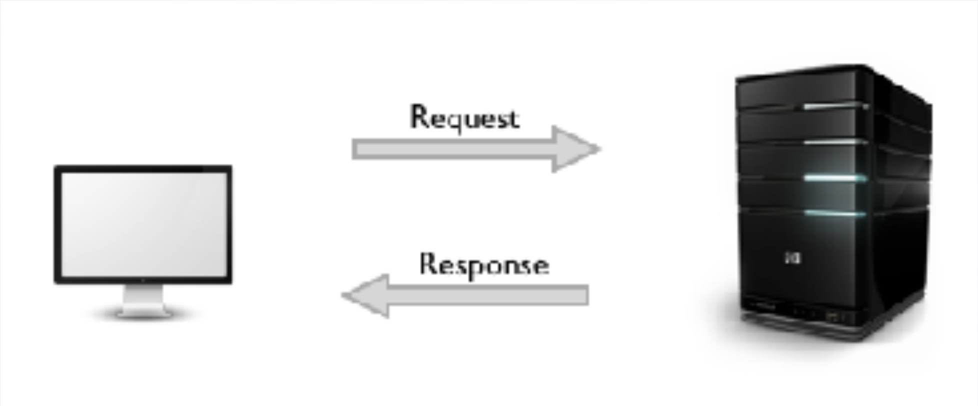
ARCHITECTURAL CONSTRAINTS

- **Client Server** - separation of concerns
- **Stateless** - can it survive a server restart?
- **Cacheable** - if data hasn't changed since last request, why recalculate?
- **Layered System** - client can't tell if connected to end server or intermediary
- (optional) **code on demand** - temporarily extend client functionality (e.g. JS)
- **Uniform Interface** - URI identifies resource that can be manipulated (verbs) & represented (mediatype) in different ways. Self-descriptive & HATEOAS (fixed entry points the transitions through states identified by hyperlinks)

REST & THE WEB

- A RESTful system is a slightly relaxed interpretation of REST properties & constraints

- RESTful systems typically use HTTP & standard requests-responses:

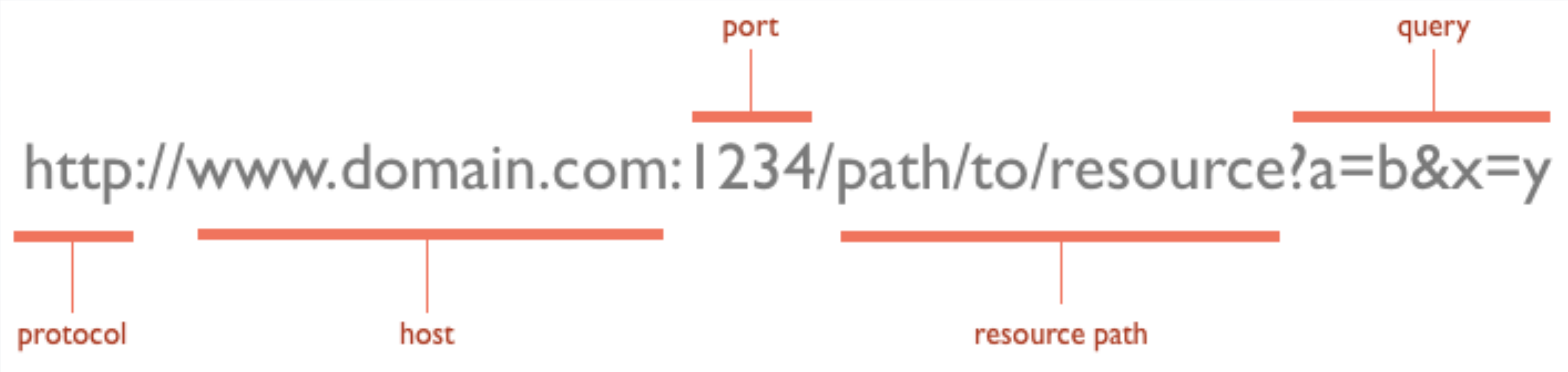


- Use the same verbs (GET, POST, PUT, DELETE, &c.) - that we have seen in labs & used (perhaps unknowingly until now) in our browsers - to retrieve web pages & send data to servers

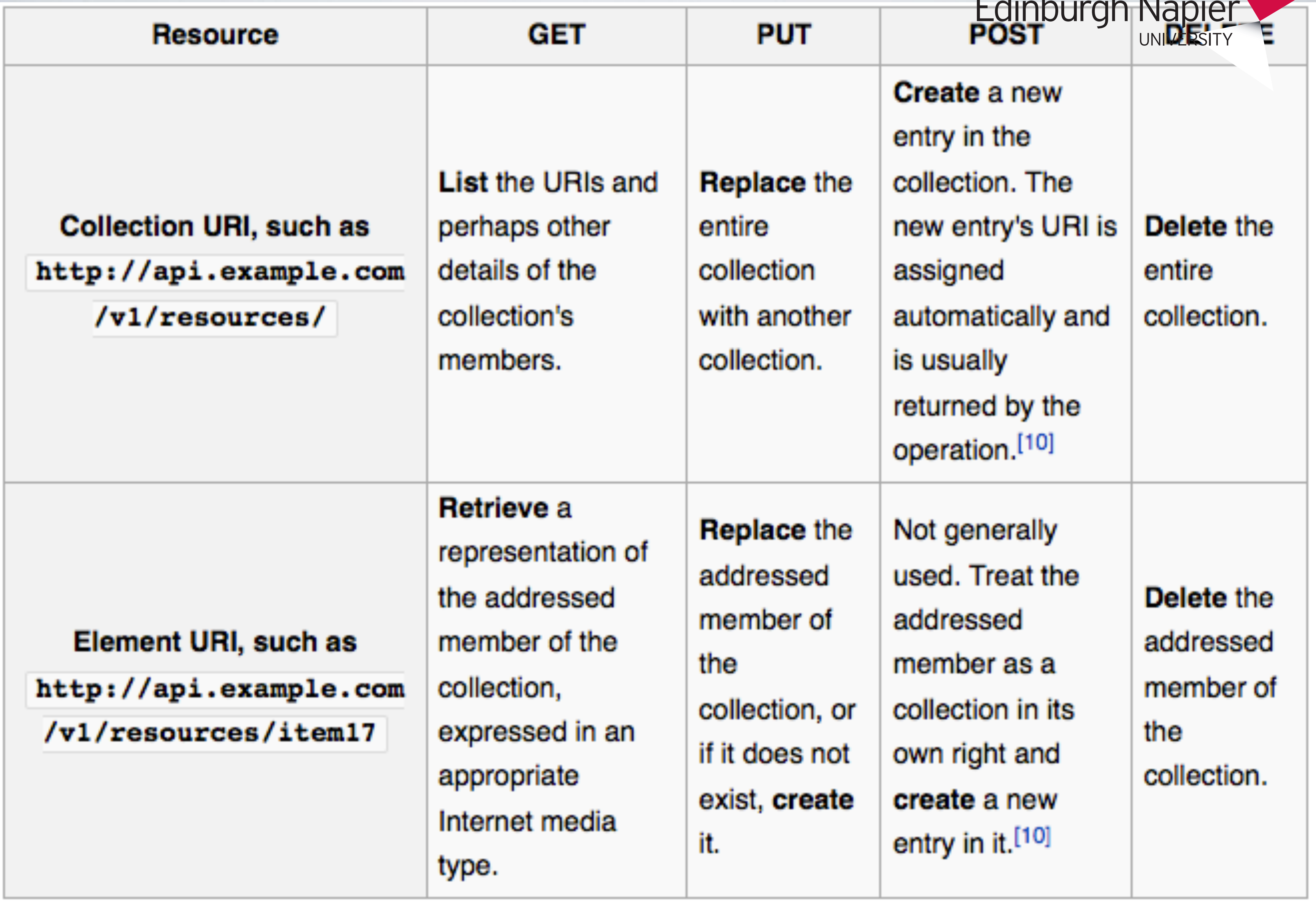
- Often RESTful systems implement some form of Create-Retrieve-Update- Delete (CRUD) system

RESOURCES, COLLECTIONS, & URLS

- Web pages are (**collections** of) **resources** - identified by a URL, e.g.
  - napier.ac.uk/students/ - would indicate a collection of student resources
  - napier.ac.uk/students/09321234 - would indicate a single student resource
- Word & naming are important in RESTful API design - Nouns rather than verbs for resources

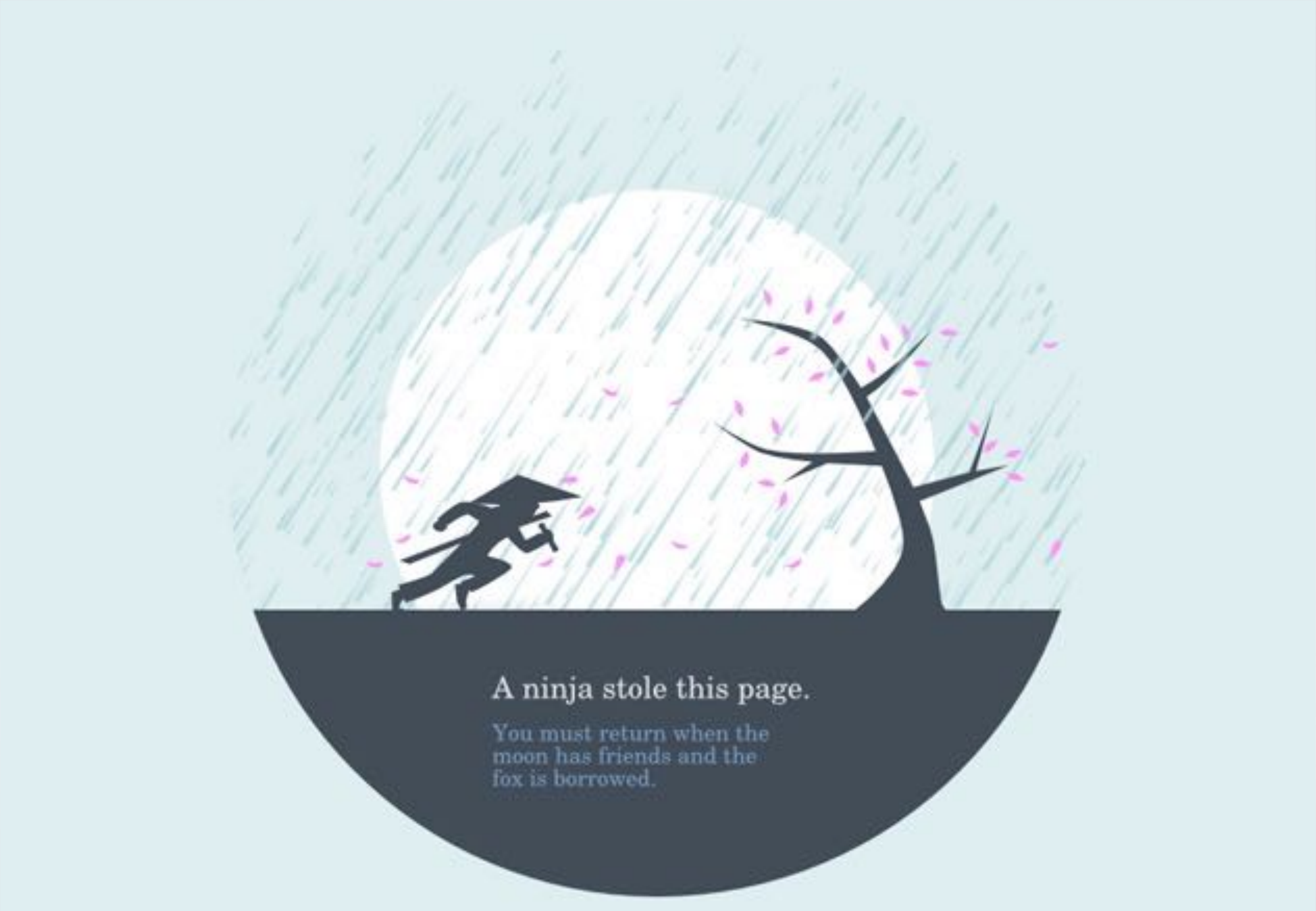


http://www.domain.com:1234/path/to/resource?a=b&x=y



Resource	GET	PUT	POST	DELETE
Collection URI, such as `http://api.example.com/v1/resources/`	**List** the URIs and perhaps other details of the collection's members.	**Replace** the entire collection with another collection.	**Create** a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.[10]	**Delete** the entire collection.
Element URI, such as `http://api.example.com/v1/resources/item17`	**Retrieve** a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	**Replace** the addressed member of the the collection, or if it does not exist, **create** it.	Not generally used. Treat the addressed member as a collection in its own right and **create** a new entry in it.[10]	**Delete** the addressed member of the collection.

RESPONSE/STATUS CODES


A ninja stole this page.
You must return when the moon has friends and the fox is borrowed.

HTTP STATUS CODES

- Make request to a server, server processes request, returns response & status code

Request + Verb > Response + Status Code

- Status Code is important - it tells us how to interpret the response from the server

- 1xx - Informational
- 2xx - Successful
- 3xx - Redirection
- 4xx - Client Error
- 5xx - Server Error

REAL WORLD APIS

- Programmable Web API Directory

  http://www.programmableweb.com/apis/directory

- Programmable Web API Dashboard:

  http://www.programmableweb.com/apis

SUMMARY

- We should now…
  - Understand the relationship between APIs & the Web
  - Know about the REST architectural style
  - Be able to consider the design of a URL hierarchy in terms of the collections of data that it exposes

NEXT

- Data:
  - How to structure it
  - How to store it